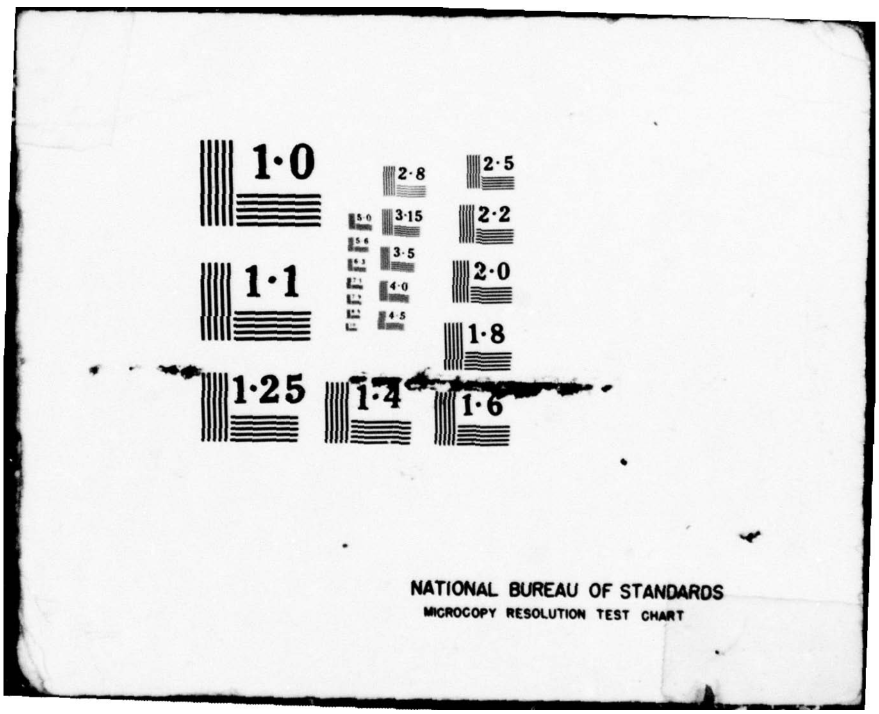
AD-A069 804 NATIONAL RESEARCH COUNCIL WASHINGTON DC COMMITTEE ON--ETC F/G 9/2
OPERATIONAL SOFTWARE MANAGEMENT AND DEVELOPMENT FOR U.S. AIR FO--ETC(U)
1977 F44620-76-C-0004

UNCLASSIFIED NL

1 OF 1
AD
A069804

END
DATE
FILMED
7-79
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

Operational Software Management and Development for U.S. Air Force Computer Systems

Air Force Studies Board

Assembly of Engineering

DDC PROCESSING DATA

II

LEVEL

PHOTOGRAPH

THIS SHEET

INVENTORY

RETURN TO DDA-2 FOR FILE

AD A069804

DDC ACCESSION NUMBER

Operational Software Management and Development for U.S. Air Force Computer Systems

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

Accession For	
NTIS GRA&I	[X]
DDC TAB	[ ]
Unannounced	[ ]
Justification Per: DDC FL-88 (B77-1745)	
By on file	
Distribution/	
Availability Codes	
Dist. A	Avail and/or special

DISTRIBUTION STAMP

DDC
RECEIVED
JUN 12 1979
D

DATE ACCESSIONED

79 06 11 022

DATE RECEIVED IN DDC

PHOTOGRAPH THIS SHEET

RETURN TO DDA-2

Operational Software Management and Development for U.S. Air Force Computer Systems

A Report by the
Committee on Operational Software Management and Development
of the
Air Force Studies Board
Assembly of Engineering
National Research Council

NATIONAL ACADEMY OF SCIENCES
Washington, D.C. 1977

NOTICE

The project that is the subject of this report was approved by the Governing Board of the National Research Council, whose members are drawn from the Councils of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine. The members of the committee responsible for the report were chosen for their special competences and with regard for appropriate balance.

This report has been reviewed by a group other than the authors according to procedures approved by a Report Review Committee consisting of members of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine.

This report represents work under Contract F44620-76-C-0004 between the National Academy of Sciences and the United States Air Force.

Printed in the United States of America

AIR FORCE STUDIES BOARD

*Brockway McMillan (*Chairman*)
Vice President, Military Systems
Bell Telephone Laboratories, Inc.

Robert W. Cairns
Executive Director
American Chemical Society

John K. Galt
Director, Solid State Sciences Research
Sandia Laboratories

*George E. Mueller
Chairman & President
System Development Corporation

+Brian O'Brien
Consultant
Woodstock, Connecticut

Frederic C. E. Oder
Vice President & General Manager
Lockheed Missiles & Space Co., Inc.

+/*Oswald G. Villard, Jr. (*Member Emeritus*)
Professor
Stanford University

Willis H. Ware
Member, Corporate Research Staff
The Rand Corporation

Kenneth S. McAlpine (*Executive Secretary*)
Debra A. Tidwell (*Secretary*)

+Member of the National Academy of Sciences
*Member of the National Academy of Engineering

COMMITTEE ON OPERATIONAL SOFTWARE MANAGEMENT AND DEVELOPMENT

W. Gordon Heffron, Jr. (*Chairman*)
Bell Telephone Laboratories, Inc.

Richard P. Case
IBM Research Center

Eldon R. Mangold
TRW Systems Group

George E. Mueller
System Development Corporation

John B. Munson
System Development Corporation

Richard P. Parten
NASA Johnson Space Center

STUDY GROUP MEMBERS

W. Gordon Heffron, Jr. (*Director*)
Bell Telephone Laboratories, Inc.

Richard H. Battin
The Charles S. Draper Laboratory, Inc.

Robert W. Berri
Aerospace Corporation

Barry W. Boehm
TRW Systems Group

Richard P. Case
IBM Research Center

Vinton G. Cerf
Stanford University

Thomas H. Crowley
Bell Telephone Laboratories, Inc.

Paul F. Glaser
Transaction Technology, Inc.

Brockway McMillan
Bell Telephone Laboratories, Inc.

Eldon R. Mangold
TRW Systems Group

John H. Manley
The Johns Hopkins University

George E. Mueller
System Development Corporation

John B. Munson
System Development Corporation

Richard P. Parten
NASA Johnson Space Center

Alan J. Roberts
Mitre Corporation

Robert D. Royer
Bell Telephone Laboratories, Inc.

PARTICIPANTS, PRESENTORS AND VISITORS

Walter S. Attridge
Mitre Corporation

Laszlo Belady
IBM Corporation

M. Leonard Birns
Aeronautical Systems Division, Wright-Patterson Air Force Base

Norman W. Briggs
Mitre Corporation

James J. Canaday, Lt. Col.
Aeronautical Systems Division, Wright-Patterson Air Force Base

William E. Carlson
Defense Advanced Research Projects Agency, (IPTL)

Barry C. DeRoze
Office of the Assistant Secretary of Defense, (I&L)

Robert B. Doane
Electronic Systems Division, Hanscom Air Force Base

Thomas O. Duff, Col.
Electronic Systems Division, Hanscom Air Force Base

Robert L. Edge. Maj. Gen.
United States Air Force, Andrews Air Force Base

Frank T. Emma, Col.
Electronic Systems Division, Hanscom Air Force Base

Michael S. Freeman, Capt.
Electronic Systems Division, Hanscom Air Force Base

John L. Funkhauser, Lt. Col.
Air Force Systems Command, Andrews Air Force Base

Sherwood C. Gordon, Maj.
Air Force Systems Command, Andrews Air Force Base

John T. Holland, Lt. Col.
Electronic Systems Division, Hanscom Air Force Base

Daniel C. Holtz
Aeronautical Systems Division, Wright-Patterson Air Force Base

Thomas H. Jarrell, Maj.
Air Force Systems Command, Andrews Air Force Base

A. Melanson
Electronic Systems Division, Hanscom Air Force Base

F. Robert Naka
United States Air Force, Andrews Air Force Base

James C. Naylor
Mitre Corporation

Eric B. Nelson, Lt. Col.
Aeronautical Systems Division, Wright-Patterson Air Force Base

Edward H. Newman
Space and Missile Systems Organization

William W. Patterson, Maj.
Air Force Systems Command, Andrews Air Force Base

M. Steven Piligian
Electronic Systems Division, Hanscom Air Force Base

Donald P. Rozenberg
IBM Corporation

William L. Schlosser, Col.
Space and Missile Systems Organization

CONTENTS

PREFACE

The Woods Hole Summer Study on Operational Software Management and Development was conducted at the Fenno Carriage House (Quissett Campus), Woods Hole, Massachusetts, during the period August 2 - 20, 1976. The study was initiated as a result of several reviews by the Air Force Studies Board (AFSB) of related problems of interest to the Commander, Air Force Systems Command (AFSC).

Over the past year, in the AFSB's review of these problems, a Steering Committee of the Board consisting of Drs. McMillian (Board Chairman), Mueller, and Ware, in conjunction with the Commander, AFSC, has selected the studies to be undertaken. A study of "Operational Software Management and Development" was held to be most useful and desirable. Thus, a Committee, consisting of Messrs. W. Gordon Heffron, Jr. (Chairman), Richard P. Parten, John B. Munson, Eldon R. Mangold, Richard P. Case, and George E. Mueller, was appointed by the Chairman of the National Research Council to accomplish this task.

The following report represents the results of the Summer Study. A listing of references is included. These consist of "Presentations to the Summer Study" and "Reference Material available to the Study Group." The presentation papers may be obtained by request to the office of the Air Force Studies Board. The referenced papers may be obtained in the customary way -- by communication directly with the authors -- or from the United States Air Force or the Department of Defense.

Many thanks are due those associated with the study for their enthusiasm and willingness.

Much credit is owed to Majors Marchiando, Patterson, Jarrell, and Gordon and their associates at Headquarters, AFSC, for providing technical cooperation and support for the necessary preliminary planning of the study.

1.0 INTRODUCTION AND SUMMARY

The 1976 Summer Study of the Air Force Studies Board (Fig. 1) was concerned with operational software management and development as this applies to the US Air Force's embedded computer resources, such as for weapon systems, in contrast with the more usual software operations of automatic data processing systems.

The purpose of the Study was not to review USAF management of specific systems, but rather to examine the ways the USAF might become generally more effective in operational software. For the USAF has found, as have many private organizations, that operational software is risky -- in cost, in schedule, in ability to perform the intended functions effectively, and in ability to evolve along with the appropriate functions.

As the Air Force increases its reliance on computers and computer techniques, it needs to take full advantage of the state of the art and the experience of others, particularly in the best use of management, development, and support of software techniques. The sums spent today by the Department of Defense (DoD) on software are considerable. DoD studies[1] reveal expenditures of more than $3 billion annually. This suggests two things: that computer hardware/software technology offers such attractive benefits that this approach is increasingly being chosen, and that improvements in effectiveness in the development, operations, and maintenance of operational software could lead to important savings.

Two aspects of software management are intertwined in this study. One concerns risk reduction. The major recommendation of this report is concerned with putting development and life-cycle costs and schedules on a sound basis, so that the USAF can commit, budget, manage, and deploy such systems in a more orderly and predictable way. The other concern is for the actual costs and schedule intervals in order to make these smaller and shorter, as well as to be realistic in doing so.

The study comprised three parts. First, USAF and DoD speakers (Fig. 2 and References) reviewed what is now done and what is planned for the future. Second, presentations were made as to what nonmilitary organizations (including military contractors) do in their software efforts. And third, a week was devoted to discussion, to distill what has been presented and studied, to draw upon individual experience and expertise, and to formulate the recommendations in this report.

Figure 1

AIR FORCE STUDIES BOARD
1976 SUMMER STUDY

OPERATIONAL SOFTWARE MANAGEMENT AND DEVELOPMENT

TOPICS

- **Contract Structure and Activities**
- **Development Practices and Tools**
- **Validation and Verification**
- **Personnel**
- **Technology Base**

Figure 2

SUMMER STUDY AGENDA

MILITARY AND DOD PRESENTATIONS

- The B-1 Program, Software and Avionics Software - J. J. Canaday
- Example of B-1 Avionics Software Development - D. C. Holtz
- The 485L Program Software - T. O. Duff
- The Management of Computer Resources in Systems - S. C. Gordon
- Personnel Practices - T. H. Jarrell
- AFR 800-14 Volumes I and II: *Computer Resources in Systems* - S. C. Gordon
- Hq AFSC Program Management Assistance Group (PMAG) - W. Funkhauser
- PMAG Charter and Method of Operation - W. Funkhauser
- ADP System Acquisition Management at ESD/AFSC - F. T. Emma
- Defense System Software Management Program Overview - B. C. DeRoze
- DARPA - Information Processing Techniques Office Projects - National Software Works - W. E. Carlson
- Mission of USAF ACS/Computer and Communications Resources Office - R. L. Edge
- Computer Aided Requirements Analysis (CARA) - A. Melanson

Figure 2A

SUMMER STUDY AGENDA (CONT.)

COMMERCIAL AND MILITARY CONTRACTOR PRESENTATIONS

- Space Shuttle Orbiter Avionics Software Management - R. P. Parten
- Citicorp Bank - Consumer-Retail Network - P. F. Glaser
- Software Testing, Validation and Verification - E. Mangold
- The Safeguard System Software Testing - T. H. Crowley
- Some Software Tools Used to Develop Other Software - L. W. Drane
- Software Development of No. 4 Electronic Switching System - R. D. Royer
- The System Development Corporation Software Factory - J. D. Munson
- Computer Networking - V. B. Cerf
- The Evolution Dynamics of Large Programs - L. A. Belady
- Programming Quality and Productivity - R. P. Case
- Software Reliability: Measurement and Management - B. W. Boehm

The range of personnel in the Study Group is suggested in Fig. 3. Many were experienced in military procurements, and many came from commercial organizations. The committee believes this variety of experience was beneficial.

Several previous activities[2], similar in intent to the Summer Study were not reviewed. Only two current USAF programs were discussed, and those but briefly. More was not possible within the time available. We believe, however, that more such information would have led chiefly to more detail in the recommendations in this report and not to changes in the fundamental concepts.

We do not mean to slight the previous studies or the general progress the USAF has made, particularly as measured by actual accomplishments. No easy way will ever be found to manage this difficult discipline, most particularly because it is continually changing. While each study may lead to improvements, as we believe this present study can, one straightforward observation is that future studies will also be well worthwhile.

This report is organized about five major topics:

- Contract Structure and Activities
- Development Practices and Tools
- Validation and Verification
- Personnel
- Technology Base

Much of the discussion is in the section headed Contract Structure and Activities. Current USAF practices and problems were examined, and a method is recommended that is intended to alleviate many of the difficulties and to reduce markedly the risks involved in software procurement. The section on Development Practices and Tools, Validation and Verification, and Personnel discusses ways of making work and management more effective, and the one on Technology Base concerns the future and recommends that the USAF should support several appropriate activities.

In summary, the committee offers seven major recommendations, although the body of this report contains many other recommendations that reinforce and extend these, as well as more discussion of the reasoning behind them.

Accordingly, we find that too often the USAF commits money and schedules to a development contract before all the ramifications of the system are sufficiently understood. To improve this, we recommend that:

The development contract be preceded by a separate contract for detailed system design requirements and no commitment be made to development until the design requirements are completed. The detailed system design requirements work may often be performed usefully by

Figure 3

SUMMER STUDY MEMBER AND PARTICIPANT AFFILIATIONS

MEMBERS

Bell Telephone Laboratories, Inc. (2)

The Charles Stark Draper Laboratory, Inc. (2)

Aerospace Corporation

***TRW Systems Group* (2)**

IBM Corporation

Stanford University

Johns Hopkins University, Applied Physics Laboratory

System Development Corporation

Johnson Space Center, National Aeronautics and Space Administration

MITRE Corporation

Figure 3A

SUMMER STUDY MEMBER AND PARTICIPANT AFFILIATIONS (CONT.)

EX-OFFICIO TO THE AIR FORCE STUDIES BOARD

Bell Telephone Laboratories, Inc.

System Development Corporation

PARTICIPANTS

Assistant Secretary of Defense (I&L)

USAF Office of Chief of Staff

USAF Assistant Chief of Staff/CCR

Hq. AFSC (3)

AFSC/Electronic Systems Division (1)

AFSC/Aeronautical Systems Division (2)

AFSC/Space and Missile Systems Division (2)

IBM Corporation

competing contractors on a level-of-effort contract. (Section 2.3 provides more detail.)

Experience teaches that often too much is attempted at once in a development. To mitigate this problem, we recommend that:

Wherever possible, the development should proceed through a series of deliveries. Each should stand by itself, not require a rework of its predecessors, and each should be of tangible use. During this period, strong feedback is needed from the using organization and the maintaining organization to the development activity, in order to correct any recognized problems. (Section 2.3 elaborates upon this recommendation.)

We find potential conflict between the role of a System Program Office (SPO) as a development organization and the life cycle success of a system. To reduce this, we recommend that:

The System Program Office, especially during the detailed system design requirements phase, be (1) responsible also for system engineering, (2) empowered to make tradeoffs within the system to optimize it, and (3) charged with balanced consideration of the system's development and life cycle. To improve the life cycle aspects, the using and the maintaining organizations should participate formally and continually in both the detailed system design requirements and the development phases as part of the SPO. (Section 2.3 bears upon this also.)

Available software tools have already provided improved productivity and quality, and moreover, show every indication of increasing both productivity and quality. Thus, we recommend that:

The USAF continue to advance the state of software tools through continued research, development contracts, and support of and coordination with the DoD Software Management Steering Committee in its efforts toward standardization of higher order languages. (Section 3.0.)

The USAF lacks a sufficiently numerous cadre of software managers. Even so, present practices in duty assignments often lead to detaching or assigning personnel at times unrelated to SPO activities. To improve this, we make the following recommendation:

To establish continuity and accountability, the period of assignment of SPO officers should be related to SPO activities. For junior officers, the assignment should span the whole job, regardless of interim promotions. To help such officers, the USAF needs to develop short courses on management procedures, on how large computer systems work, and on management case histories. (Section 5.0.)

The technology base is in great ferment for computer systems.

Therefore, the USAF needs to cope better with the present and prepare for the future. To this end, we recommend:

The USAF maintain a vigorous program to improve the technology base. Among the topics needing attention:

o The structure of, and quantitative measurement of, development processes.

o Extending present software development tools and adding new ones, such as validators for requirements and design.

o Computer hardware and software architectures to improve system performance and to extend life cycle utility.

o The applications of mini- and microcomputers, including problems of logistics, reliability, and maintenance, as well as function and performance.

o Data communications, input and output methods, and means for dealing with interfaces between systems. (Section 6.0 concerns this recommendation.)

As our final recommendation, we find that the DoD is attempting, through Directive 5000.29 and its Software Management Committee, to advance software management and the utility of computer systems to the agency. We recommend that:

The Air Force Systems Command strengthen its support of the participation in this committee by the USAF, providing a stronger focal point for its own activities to promote such support. The present organization, the Directorate of Computer Resources Development Policy and Planning (XRF), in AFSC, does not call for the rank, staff size, or charter that such a focal point organization requires.

While software development should be less difficult to manage in the Air Force when these recommendations are implemented, it will still not be easy. Even so, the potential benefits of computer systems are so great that the opportunities are continually extended. Coupling this with the rapid evolution of computer hardware and software makes for a continual management challenge, intensified for software by its conceptual rather than concrete nature. For the USAF, the challenge is increased by procurement processes. Such processes emphasize the buyer-supplier relationship, even with developers, and their historical basis lies in the procurement of systems not significantly related to computers or software. The challenge will remain and will be worth continued effort.

2.0 CONTRACT STRUCTURE AND ACTIVITIES

This major section first discusses the present system for development, emphasizing the problems. Then certain principles that seemed important to the study group are discussed, and finally a new approach to development of software is recommended and discussed.

2.1 PRESENT PRACTICES AND MAJOR PROBLEMS

Once a program is conceived, it takes its first official life in what is called a Required Operational Capability (ROC) document. For major systems, this is reviewed and approved (or not) by Headquarters, USAF. For smaller systems, other organizations accomplish the same function: to determine that the ROC is a valid requirement. On approval of the ROC, a Program Management Directive (PMD) is prepared, and the requirement validated and brought to a System Specification level. The Management Review I then occurs, using the ROC and the preliminary system specification.

Between Reviews I and II, work continues to validate the concepts and to decide how to implement them. A procurement package is prepared. Management Review II then reviews and confirms the full scale development decision.

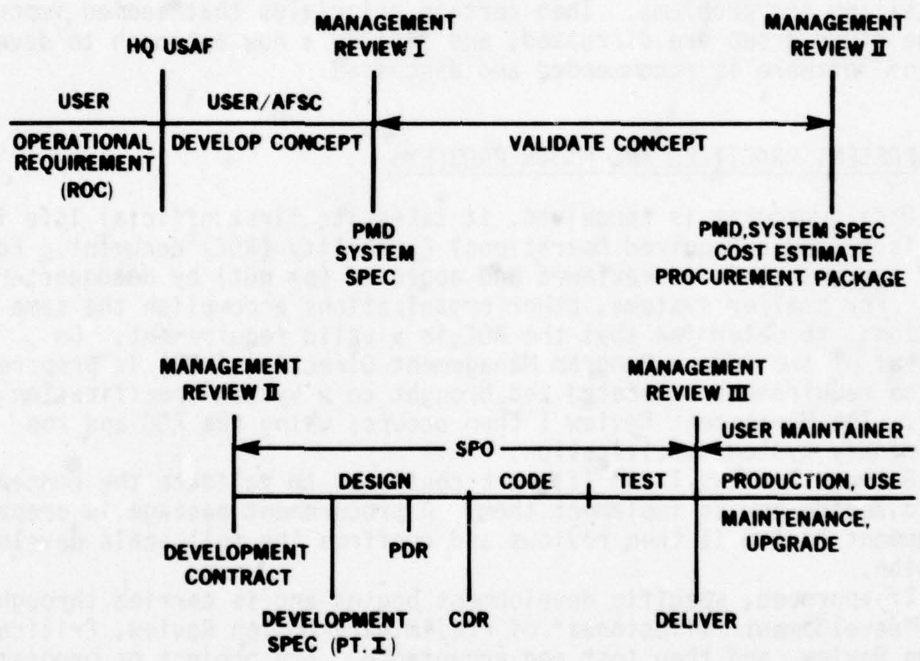
If approved, specific development begins and is carried through the major development "milestones" of Preliminary Design Review, Critical Design Review, and then test and acceptance. The project or program then moves to the Management Review III, called the Production Readiness Review. Upon approval, the appropriate quantities of systems are then purchased, deployed, operated, and maintained.

For software itself, Fig. 4 shows the activities more specifically. The interpretations there are that the user organization (which wants the capabilities described in the ROC), and the USAF Systems Command, bring, in the PMD and System Specification, a description of what is wanted. This is improved in detail and the improvement is reviewed at the Preliminary Design Review. Design is then completed, reviewed again, the program coded and tested, and then delivered.

Since software is replicated so easily, there is often no production phase for it. After delivery, the activities are of two types -- use and maintenance. Of course, "maintenance" is somewhat of a misnomer

Figure 4

OVERSIMPLIFIED PRESENT DEVELOPMENT PROCESS



when applied to software. There are simple errors to be corrected, blunders clearly contrary to intent, an activity properly called maintenance, but "maintenance" also includes coping with errors which are due to design flaws, either a poorly stated or missing requirement, or a poorly designed and coded response to the requirement. "Maintenance" also includes upgrading of the system, the creation of new capabilities. Such work is actually development, but it is called maintenance because it is performed by the organization responsible for maintenance, and budgeted appropriately for that organization.

Three particular problems occur with this approach (Fig. 5). One is that the development generally is entered upon before the job is fully understood. The ROC, the PMD, and the system specification generally stress what we call the primary mission, to the neglect of details, of how anomalous and extreme conditions are to be treated, and of collateral tasks. This is only human, but in many software programs, it is the anomalous conditions and the collateral tasks which end up determining the actual amount of work to be done and its cost and schedule. This is especially true of what Everett[3] calls "policy" software, but is true also for "physical principles" software. Examples include what to do if the input data, say from sensors, are stale or missing; what to do if keyboard operators input unrealistic data; whether or not to keep track of activity by user, by device, or whatever, to determine how close to capacity the system is operating; and whether or not to include, in the operational software, routines which trace its flow, the better to be able to diagnose problems which may occur, or later to improve efficiency.

Another major problem is the often low involvement of user/maintainer organizations during the development. The theoretical concept appears to be that the user, having stated the requirements, need not be involved during the development. Partly this concept is a reaction to the concern that strong user involvement would lead to a steady change in requirements and a resulting loss of control over the development. The contrast, in the extreme, becomes one between a tidy development with no changes but with a product largely irrelevant when delivered, and a development never brought to completion because of constantly changing goals. We believe there is an effective and necessary middle ground of user and maintainer organizational involvement. Certainly there is need for full and continued planning for the operations and maintenance activities, and of the development products which are needed for them. Beyond that, there is the filling in of details, resolution of unclear requirements, and the formal change of requirements as may become appropriate.

Greater involvement exposes also the conflicting goals of the user, maintainer, and developer organizations. The user wants maximum capability quickly, the maintainer wants a "perfect" product from the developer, and the developer has limited resources to work with. By far, however, this conflict is healthier and preferable to no interaction at all.

The third major problem, we believe, is the "turnkey" or "one-shot" philosophy of development. Each development is considered a thing unto itself, drawing little in management and expertise from previous similar

Figure 5

MAJOR PROBLEMS

- **Contract price and schedule set before job fully understood.**
  - **Concentration on primary mission; neglect details, anomalous and extreme conditions, collateral tasks.**
- **User/maintainer involvement has been too low.**
  - **Preparation for O & M, upgrade.**
  - **Requirements change during and after development.**
- **"One-Shot" development.**
  - **Attempt "everything" first time.**
  - **Dismiss expertise at delivery.**

projects, especially in the System Program Office, and dismissing the expertise created during the development while the product is still in early use and evaluation. Continuity in SPO organization and people and continuity in contractor personnel are of value. At the minimum, a "soak" period for the delivered program would profit from availability of its developers.

Dangerous also is the possible emphasis of the development phase at the expense of the system life cycle. Both types of continuity are important -- continuity of development expertise, and continuity of attitude toward the system throughout its life cycle.

While the above notes the major problems, it is useful to go into more detail about the difficulties arising from the present methods, the better to appreciate the recommendations we make later in this report. It may appear that the problems are peculiar to contracts purely for software. They are not; they are then more visible, but they and their variants appear also in fully embedded system contracts. Let us hasten to say that the problems noted do not all occur in every program. But let us state also that these are real, not hypothetical, problems, as is borne out directly by Refs. 4 and 5, indirectly by Refs. 6 and 7 (which discuss why two specific programs were successful), and by the Study Group's personal experiences.

2.1.1 Requirements

Figure 6 lists some problems with requirements. Some get stale because users are not continually involved, because the change process is cumbersome, and because changes in requirements cause changes in costs and schedules and, ipso facto, are best avoided. Further, the time from initial ROC generation to full scale development may simply be too long.

Contributing to the high risk in software cost and schedules is the general incompleteness of the requirements -- a concentration on the primary mission to the neglect of anomalous and extreme conditions, of interfaces (especially those with other systems), and of collateral functions. It is true that the discovery of some problems will not occur until the specifics of the coding are nearly complete, in part because the problems arise out of the implementation method, but it is not true that this is the proper time to discover and resolve design problems. And it is true that it is difficult to get some parts of the user/maintainer organizations involved during the requirements phase of a project, when the actual use and maintenance will not occur for some time (especially when the personnel may well change during the interval), and when current duties and problems are pressing. Perhaps software is different from hardware in that, with software, so many decisions, which truly shape the system, have to be made so early, and cannot be postponed. Whether or not this is an important distinction of software, the fact remains, and getting the necessary degree of involvement is often difficult.

Incompleteness spills over into life cycle and total system aspects.

Figure 6

MORE SPECIFIC PROBLEMS

REQUIREMENTS

- **Get stale: change process cumbersome.**
- **Incomplete as to:**
  - **Anomalous and extreme conditions.**
  - **Interfaces, especially as with other systems.**
  - **Collateral functions.**
- **Poor on life cycle, total system aspects.**
- **CPCIs, not total system plan.**
- **Realism often unknown.**
  - **Risk analysis lacking.**
    Within the system.
    Interfaces with other systems.
- **Hardware/software tradeoffs already done.**
  - **Hardware committed too soon, especially memory size, peripheral and microcode processing, new technology.**
  - **Software expected to "fix" hardware.**
- ***Adjectives, not numbers.***

Without forethought and planning before and during development, the operations and maintenance phase, generally the longest and most expensive phase of a system, can be severely crippled, made less effective and more expensive.

On the other hand, the wrong type of planning can break a system into contractual parts which may not add back into a coherent whole system. Contracts must inherently call for specific products -- Computer Program Configuration Items (CPCIs) for software. A proper balance must be struck which enumerates properly all the elements while leaving room for flexibility.

Realism in requirements is also often a problem. "Can the hardware/ software perform critical tasks effectively?" is often not answered early enough. The "top-down" structure concept of programming has the attribute that it could be used to emphasize the structure of a program at the neglect of such details. Interface realism is often difficult to work out, especially with other systems. The other systems may have their own SPOs, with their own problems, or may, if already deployed, not even have a SPO to coordinate and work out the interface implementation. Between services, the problem may be even more difficult, especially when terminology and procedures have long-established differences. It may be difficult to resolve problems of realism early, but it is important to recognize potential risk areas, and to make a better informed decision as to their resolution before proceeding with a full scale system development.

Another problem with requirements is that hardware decisions are generally made before software development begins, and the software, because it is so "flexible," is left to fix any problems. (We would hope that it is now well understood that "flexibility" also means "complexity," and is difficult to develop, use, and maintain.) At the simplest level, no hardware decisions should block increased memory size (no early estimate can be precise enough to permit such a decision), nor should such a decision block the ability to use peripheral processor or microcode (firmware) methods of gaining computer power. Such changes will often make good a large investment in software development which might otherwise be scrapped. Trying to fit a job into an undersized and underpowered computer is a sure way to high cost and risk of failure.

Obviously enough, and especially in a contractual sense, it is insufficient to require that a system work "well" or "accurately," or "reliably," or "fail gracefully," or be "robust to load," to quote a few appealing adjectives. Numbers should be used, both to define the system and to permit the government knowledgeably and legally to accept or decline the delivered product.

2.1.2 Management

Figure 7 shows some of the problems related to management in which we include both people and procedures used in accomplishing a system and running a SPO.

Figure 7

MORE SPECIFIC PROBLEMS (CONT.)

MANAGEMENT

- Tour of duty not job related.
- User/maintainer participation has been low.
- Software is not identical to hardware, except in need for discipline.
- Piling up standards, procedures, documents from other jobs is not better management, nor is more fragmentation of the contract controlled work.
- Lack of Government system engineering capability.
  - Inflexible emphasis on cost, schedule, items, "policy" at risk of total system.
  - Interoperability of systems suffers.
  - Reliability and quality assurance not stressed.
  - Poor reviews both ways (government and contractor).
- Lack of a sure way to assess progress.

We recognize that the AFSC is conscious of and is reacting to the major management problem we see -- that the tour of duty of officers in a SPO is not related to the SPO's activities. Regardless of management directives and procedures, there is a personality element in management, and continuity is valuable.

Another problem, the low or nonparticipation of users/maintainer organizations during development, has already been touched on. Lack of it not only hampers accurate definitions of the requirements, but has the potential for poor coordination of the delivery-to-use transition and poor preparation for the post-delivery phase of the system..

The third and fourth points of the figure go together: "software is not like hardware" is true except in the need for discipline, and the discipline cannot be automatically provided by applying blindly anything and everything which contributed to success in other developments. We believe each project to have unique problems and challenges, requiring a unique set of specific management techniques. Techniques successful in other programs are rightly candidates and guidelines for new programs, but should not be automatically applied. Especially this is so because the development process is evolving and will continue to evolve. But surely overfragmentation of a task, as a management technique to provide greater visibility, is easily contrary to the "total system" goal, and too much required coordination with other bureaucratic organizations, too many reviews for outside or higher authorities, can easily divert the SPO's attention from its primary task. We still believe in straightforward specific assignment of authority together with responsibility as a successful management technique.

In a development, it appears to us that the job of the SPO is not simply to manage its contracts, but rather to ensure that the product is proper as a system: in short, the SPO carries the systems engineering responsibility -- not headquarters, not the user/maintainer, and not a possible systems engineering contractor. A SPO can delegate only authority, but not responsibility.

Lack of this attitude and authority puts the SPO in a poor posture. It must insist, until it proves impossible, on contractual items, on cost and schedule, and must enforce "policy" to do this, at the risk of the whole system. Interoperability with other systems will suffer unless the initial definition proves perfect, and reliability and quality assurance will go the same way. There may thus be successful reviews, but successful only in that they were held on schedule, not because the subjects addressed were understood by both the contractor and the government, as their content would affect the total system. We believe a proper review ought to be easily conducted, but because the content has been previously covered so thoroughly that the review becomes primarily a formal approval of well-understood decisions.

2.1.3 Procurement

The weight and volume of the Armed Services Procurement Regulations have become so large that it must be extraordinarily difficult to create a contract which is right for the job and still in full compliance with the Regulations. Our arguments are with more fundamental matters, however.

Given our belief that the development contract is generally consummated before the system to be developed is sufficiently understood, it follows easily (see Fig. 8) that lowest price as a criterion for contract award will fail. For too low a price, the government will get too little effort, an improperly short schedule (schedule thus also being a poor criterion in such a case), inadequate staffing, and possibly insufficient facilities. Contractors must bid to supply no more than the specifications call for and be optimistic of their success and of the specification's completeness. Changes in scope can be used to make up deficiencies, but we are convinced that a job which starts off badly has little chance of becoming right through the change-of-scope process.

Our discussion brought out cases in which the government believed jobs to be bigger than the proposals tendered, and was later proved to be correct, but had no way to prove this at the time of bidding and to disqualify the bidders. Especially with a fixed price bid, this is a delicate problem.

Part of the procurement problem is in the "turnkey" concept, that the developer is to "get in and get out" and simply to fulfill the specifications on time and within cost. This approach tends to set aside life cycle costs and performance, and is unresponsive to changing requirements. Having a SPO exist for only one procurement does nothing for related procurements, and using the development contractor for only the development does little for the early life of the product or for related following developments. Noninvolvement of the software contractor in hardware, in computer power and memory size decisions, is also an often troublesome factor in the "turnkey" approach.

Finally, procurement practices are slow, because the practices are legalistic and are concerned with tidiness as opposed to "useful system" goals. Engineering Change Proposals, Changes in Scope, cause cost and schedule changes and are thus discouraged, and an adversary attitude is created between government and contractor, to the detriment of the final product.

2.1.4 Validation and Verification (V&V)

These words (Fig. 9) have achieved such generic application that they have become almost useless in defining a specific phase in software development. Originally (for us, at least), they were used to denote the delivery and acceptance test phase, the final part of development where the product was shown to be in conformance with the specification, and, perhaps additionally, usable by the government with confidence.

Figure 8

MORE SPECIFIC PROBLEMS (CONT.)

PROCUREMENT

- **Price criterion for contract award fails:**
  - **- Poor spec causes underbid, underschedule.**
  - **- Underbid causes poor staffing and facilities, changes in scope, overruns, late-----.**
  - **- Bidders must bid minimum job (and quickly) to win.**
  - **- USAF lacks good ways of rejecting underbids.**

Figure 8A

MORE SPECIFIC PROBLEMS (CONT.)

PROCUREMENT

- "One-Shot" or "Turnkey" development causes:
  - Poor life cycle costs and performance.
  - Unresponsiveness to changing requirements.
  - Present methods inhibit valuable continuity in USAF and contractor people, especially from one job to the next.
  - Hardware first, software catch up.
- Software may be a nearly "invisible" part of a total system procurement
- Processes are slow.
  - Stale requirements.
  - ECP (Within the design, and for changing requirements) discouraged.
  - Adversary vs. useful-system-goal oriented.

Figure 9

VALIDATION AND VERIFICATION

- No common definition exists, is probably impossible.
- Too often confused with system engineering.
- Too often an add-on at end, especially with independent V&V contractors.
- "More is better" may be true, but V&V is not a catch-*all*-errors technique. R&QA emphasis in design and code is a higher payoff way to spend money.
- "Once done, it's over" is false.

Since then, it has been realized that requirements, specifications, and all other important decisions and statements about the system must also be validated and verified -- found to be true, appropriate, etc. and, hence, the confusion as to precisely which contract phase and activity is meant by V&V.

We would prefer that this latter activity be rightfully recognized as part of system engineering (the importance of which is noted elsewhere), and the usage of "V&V" be reserved for testing of code.

Since the purpose of testing is to catch errors before it is too late, it easily appears to be a good place to spend money. In fact, it may cost 100 times as much to fix an error found late than to correct a faulty requirement,[8,9] and balance between system engineering and final testing, as well as other parts of the work, is necessary. It is better to prevent an error or discover it early than wait for testing to find it.

In fact, final testing is not the last chance to find errors. Some errors will slip through, to be found in operations. Most operational programs do not include debug capacity -- and the analysis capability off-line to use such information -- and this we believe to be a flaw.

2.1.5 System Design and Development

Figure 10 indicates some problems we feel exist in present development practices.

"Core of design lacks life-cycle orientation" is one way of saying that a computer program can be written several ways, and that the better its life cycle aspects are defined and understood, the better the design can fulfill the life cycle needs. For example, an "information management" system could have at its core an easily defined and efficient "data base management system" and then give the user full capabilities to define data structures, input formats, processing output displays, and control over communication channels for input and output, the better to meet the evolving needs. Thus, the development could be more of a user tool than an operational system.

If such a system could be foreseen to grow, then the capability to grow could be planned in. Memory growth could be made possible, performance and debug monitors provided intially, substitution of new hardware made possible by planning either upward compatibility of code, or planning for portability of the code to another superior computer, or by making hardware/software tradeoffs more possible to move toward more distributed processing or multiprocessing.

We believe that development can be made more effective and efficient by the use of what are called tools. Software development in the past has been "labor intensive," and includes much work that is considered drudgery and which, therefore, is not always done thoroughly. Tools which affect design analysis, code development and tests, system test, and operations, maintenance and upgrade have all been poorly treated in many contracts, both as to their usage and as to their cost and status as deliverable items.

Figure 10

HARDWARE/SOFTWARE SYSTEM DESIGN AND DEVELOPMENT

- Core of design lacks life-cycle orientation.
  - Changes in requirements, technology.
  - Memory growth capability.
  - On-Line, in-use debug capability.
  - Hardware/software tradeoffs.

Figure 10A

HARDWARE/SOFTWARE SYSTEM DESIGN AND DEVELOPMENT (CONT.)

- Tools lightly treated by contracts.
    - Very important for:
        - Design analysis.
        - System test
        - Operations, maintenance, upgrade.
    - Pricing in contract, GFE tools are problems.
- Data interfaces, structure lightly treated.
    - Intra-and inter-systems.
- Higher order language control need improvement.
    - Especially for military languages.
    - Especially for life cycle aspects.

Computer programs are written as modules that perform operations on data supplied to them, and, in turn, supply data to others. The interfaces between them are data: only recently have development techniques been created which reduce errors in such interfaces by better defining and organizing data structures and transfer. Contracts should require such superior treatment of these interfaces.

Interfaces between systems also are a problem, in many cases because the responsibility is difficult to assign. In many fields, notably Command, Control, and Communications (C^3), there appears to be a need for top level multisystems responsibility to resolve such problems of interoperability. No such organization seems to exist.

Higher order languages (HOL) are now considered key elements in programmer productivity and effectiveness and in maintenance during the life cycle. There has been a trend to proliferation, to specialization of such languages, to the neglect of life cycle aspects. The problem is intensified in that each HOL also implies a set of tools, facilities, and personnel training and specialization; too many HOLs fragment the maintenance effort, increase cost, and reduce effectiveness.

2.2 SOME GUIDING PRINCIPLES

Figure 11 indicates some truisms pertinent to software development.

Possibly the most important of these is the first: that the real job is much bigger than the primary mission. By primary mission we mean the concept of usefulness which is the heart of the Required Operational Capability. For example, in one Bell System message switching system, the computer program consists of three parts of equal size: message transfer, maintenance, and administration. Of these, message transfer is what we would call the "primary mission." Even so, only a fraction of that code is used when there are no problems with the message. Most of the code is involved in testing for errors, for improper data, and for dealing with these and other anomalous cases. Yet the concept of usefulness, and the code which will have by far the greatest use, is in moving an errorless message through the system. The ability to estimate the size of the total job, in the Study Group's experience, decreases rapidly as one moves further away from the primary mission leading, in some cases, to a complete omission of some collateral functions, with the degree of uncertainty greatest when the function is new to the estimators.

Considerable evidence[8,9] supports the second truism noted, that errors are best found early, with ratios of costs running to 100:1 between an error found in the operations phase after delivery, and one found when design is still in progress. The most plaguing of these errors are those in which the basic concepts and algorithms break down, when fixed point scaling is overrun (and not guarded against) or when unanticipated combinations of data cause nonsense (and possible harm) in the output, or even "crash" the system, bringing it to a halt.

The third point, small programs are surer, is also within our common experience and can be said in two ways. One is that it is much

Figure 11

SOME PERTINENT TRUISMS

- **The real job is much bigger than the primary mission. Understand the whole job before committing development money and schedule.**

- **The later the error is discovered, the more expensive the fix. Save money and time by thorough requirements and design planning.**

- **Plan the job but, when possible, don't do it all at once. Deliver first a minimum, but useful, core part, then enhance with succeeding deliveries until the job is complete.**

- **Fit the management structure to the work, to improve the work. Each job may require a different approach.**

- **No document should be produced unless its user is known and has participated in defining its contents.**

easier to add functions to a program that already works than to try to make all functions work at the same first time. Much of the thrust of new design methodology is aimed toward independent segments of code, minimizing interrelationships which, if wrong, can hurt several segments simultaneously. Most difficult is the case in which shared code is right for one segment and wrong for another: one must first recognize this circumstance before achieving a correction.

The second part of this point is that what is delivered is often less than the original intention. Many programs go through more than one "crisis," where memory available is recognized as inadequate or when timing needs cannot be met. The common solution is to reduce function, which is unwilling evidence that more than the absolute minimum was requested in the first place. We suggest that it might be better to require only that absolute minimum in the first place, and then increase function in an orderly and informed manner.

Job-oriented management methods and reader-oriented documentation have in common the motivation of programmers. Software is notorious for poor documentation. Only the code itself is a perfect documentation. Code is either incomprehensible to the reader or there is simply too much for a person to comprehend usefully. But the programmer remains emotionally involved in writing the code and making it work, and often views documentation and management visibility as distasteful diversionary tasks.

The evolving solution, it appears, is to organize the design, code, and test process better, structuring it so that documentation and management visibility occur almost automatically. That is, these become intrinsic parts of the process and not seeming diversions of effort.

Job-oriented management applies also to specifications for deliverable documentation. We are opposed to routine use of existing documentation specifications even if they have been successfully used on other contracts. Such specifications are properly candidates for use and good checklists, but should not be included as a matter of routine. The real needs should be specified: real needs are more willingly and thoroughly met.

2.3 RECOMMENDED DEVELOPMENT PROCESS

Figures 12 and 13 highlight our recommendation that a development consist of two major phases, contracted separately.

The first is called the Detailed System Design Requirements contract. Its fundamental purpose is to put the actual development on a sound footing by developing the detail in the requirements so often lacking.

To do this requires strong participation by user and maintainer organizations with the SPO. We recommend this be made a formal part of the SPO, rather than a temporary type of duty. The user participates by filling in and making decisions on the details in the requirements, by determining what will be needed from the development (training

Figure 12

RECOMMENDED DEVELOPMENT CONTRACT STRUCTURE

- **Two contracts.**
  - **Detailed system design requirements contract.**
    - **- User/maintainer strong participation: institutionalized**
    - **- Hardware/software tradeoffs included.**
    - **- Level of effort set by USAF.**
    - **- Possibly competitive. (Management of competitive efforts is difficult, requires extra effort by USAF)**
    - **- Creates solid information, bid, schedule for the actual development and for the life cycle plan.**

Figure 12A

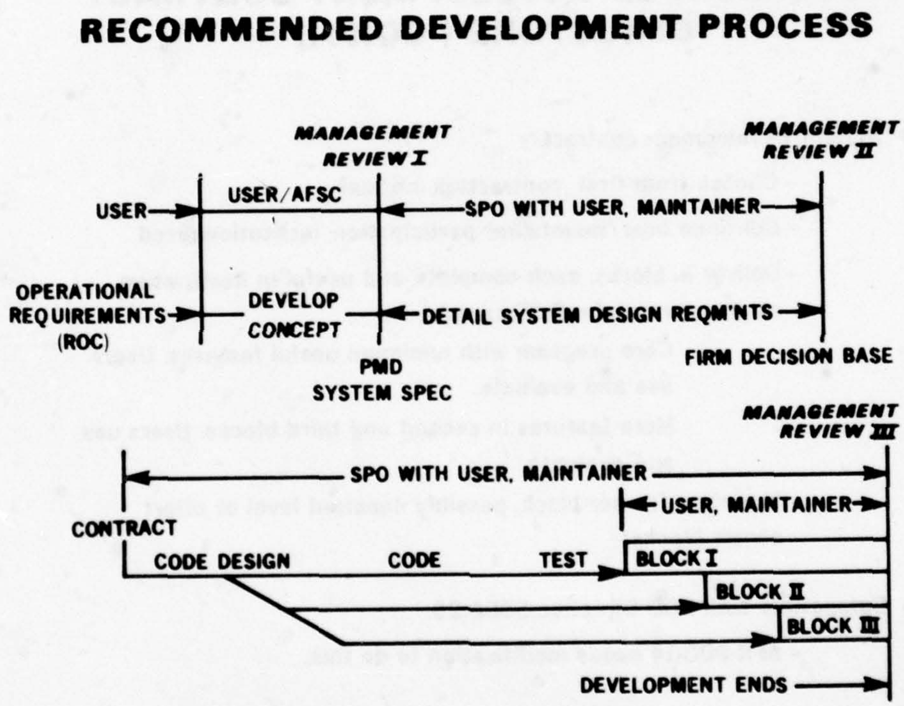
RECOMMENDED DEVELOPMENT CONTRACT STRUCTURE (CONT.)

- System development contract.
  - Choose from first contractors (critical)
  - Continue user/maintainer participation: institutionalized
  - Deliver in blocks, each complete and useful in itself, when at all appropriate. For example:

    Core program with minimum useful features. Users use and evaluate.

    More features in second and third blocks. Users use and evaluate.
  - Incentive fee per block, possibly constant level of effort across blocks.
- Compatible with DOD Directive 5000.29.
  - AFR 800-14 needs modification to do this.
- "Open Door" both ways - stress "team" and "whole system", and recognition and reconciliation of varied goals of participants.

Figure 13


RECOMMENDED DEVELOPMENT PROCESS
MANAGEMENT REVIEW I
MANAGEMENT REVIEW II
USER
USER/AFSC
SPO WITH USER, MAINTAINER
OPERATIONAL REQUIREMENTS (ROC)
DEVELOP CONCEPT
DETAIL SYSTEM DESIGN REQM'NTS
FIRM DECISION BASE
PMD
SYSTEM SPEC
MANAGEMENT REVIEW III
SPO WITH USER, MAINTAINER
USER, MAINTAINER
CONTRACT
CODE DESIGN
CODE
TEST
BLOCK I
BLOCK II
BLOCK III
DEVELOPMENT ENDS

material, test capabilities, etc.), as well as other needs (facilities, personnel, etc.) outside the development required to make operations successful. The maintainer's activities are similar, with some needs to be filled by the development and some by other sources.

In this work, a level of effort contract, at a level thought appropriate by the USAF and not influenced by lowest bid, seems appropriate. It might also be conducted in parallel by competing contractors. In such an arrangement, however, the USAF must work out better ways of managing such competition. Flow of information must be good both ways between contractor, SPO, user, and maintainer. What one contractor is told and another not can lead to legal problems. Both must know the same requirements or else different systems will result, and disputes upon final award.

We recommend that hardware/software tradeoffs be possible. Risk analysis may show more computer power or memory to be needed or that other architectures might better promote reliability, etc. Knowing potential constraints (cost, size, interfaces, etc.), the contractors should consider alternatives, the better to optimize performance within the constraints. This implies that the SPO responsibility must span the hardware and the software, must include the total system, and must include the resolution of such tradeoff studies between hardware and software. If appropriate, one contractor may do both hardware and software design and selection; if not, then the SPO must resolve conflicts and make the necessary choices and decisions.

With the output of the first contract, the USAF will have the basis for an informed decision-to-proceed. It may turn out that the cost is beyond the operational value: better to learn then than later. In either case, proceed or not, the uncertainty factor in the actual development is now less than it was in the beginning; exactly what the system will do is better known all around, and realistic requirements are sorted from the unrealistic.

The System Development Contractor should be chosen from the prior contractors. Not to do so breaks continuity, opens the system up to redesign, and renews the lowest bid syndrome discussed earlier. Having a first contractor become the system engineer, but using an entirely new contractor as the developer, is possible, but not as effective as directly using one of the first contractors. Taking parts of different proposals and thus redefining the system and the development is not recommended since it breaks continuity. The proper time to get a good system design is with the detailed system design contract. If, nonetheless, it appears best to combine elements from the work of different contractors, it must be recognized that new technical and cost proposals are then required, and an evaluation that the elements will mix successfully.

We recommend the user and maintainer continue strong formal participation during the development. There is no better way to ensure that requirements will be fulfilled and to understand all of the ramifications of the computer program than to be directly involved in the reviews. Further, since some changes are inevitable, even given the details from the first contract, the changes will be more effective and

consistent by maintaining this continuity. Changes, of course, still would be approved formally, to prevent cost and schedule effects from being hidden. This time period is also the time when the user and maintainer are making final preparations for post-delivery activities, and this is when such participation in development promotes an effective transition.

We recommend that, when possible, delivery be made in stages, what we call "blocks": each block is to fill some user needs in a tangible way, and later blocks should not cause code changes in earlier blocks. There are two major reasons for this. One is to reduce the total amount of code to be developed in the first unit, to get the core parts running sooner. Along with this is the fact that such a strategy will reenforce the need for a sound design for the total program and will better force independence between different application modules in the code. The other major purpose is to get improved and earlier user and maintainer evaluation. We do not mean the first block to be a toy, not really put to use until later blocks are delivered, but rather simply an early subset of the full capability.

It may be argued this approach will stretch out a project. We think not, and especially not in high risk projects, where problems may prevent almost any effective delivery. There are examples, e.g., the SAFEGUARD project, in which the program was very large and yet was delivered successfully as a single unit, but this is the uncommon exception and ought to be attempted only when the contractor is exceptionally competent and experienced in the field. In fact, in SAFEGUARD, there were large separate modules equivalent in many ways to our blocks, delivered internally as blocks to the testing organization, though not to the user. Intentionally phased deliveries were also made to the user, containing enhancements. One reason for such later deliveries was to exercise the processes designed into the system for adding improvements as part of the "maintenance" function.

It may also be argued that breaking some programs into blocks is artificial. If it truly is, then we do not recommend this process. But we believe such situations to be exceptions rather than the rules, even when "physical principles" are involved. In "policy" programs, the block concept should almost always be appropriate.

The block concept should also permit tradeoffs in contents of blocks, the better to accommodate problems while maintaining overall schedule. To guard against casual or arbitrary slipping of features into later blocks, we recommend an incentive fee contract structure that rewards the contractor for on-time delivery by block and function, and denies rewards when features or blocks are delayed. To implement this requires that the features to be delivered be directly tangible and demonstrable in a rather sharp yes/no manner in the development contract.

Our discussions suggested this approach is compatible with DoD Directive 5000.29, especially in that it promotes an informed Management Review II decision. It appears that USAF Regulation 800-14 needs modification, however, which we recommend (see section 2.3.4).

We stress the need, throughout the project, to inculcate a "team" attitude involving the user, maintainer, SPO, and contractor, with the

SPO as the leader. We recognize that different organizations may and do have differing goals, the user to get greatest capability, the SPO to meet schedule and budget as well as provide a good product, the maintainer to get a program easily maintained and, possibly, easily enhanced, and the contractor to merit a proper reward for good performance. This is not peculiar to software and is simply one way of defining what management is all about.

Informed readers will observe that, while our recommendations would cause changes in the present "standard" procedures, the USAF is using a multiphase approach in certain current procurements (e.g., the Joint Surveillance System). Additionally, the processes called out in DoD Directive 3200.9 of July, 1965, have somewhat the same structure as this recommendation. And, finally, others might view our recommendation as an extension of (and possibly more expensive version of) the current Validation Phase. We hesitate to agree with the word "extension", since we call for vital changes in substance in the output of this phase. In particular, the proposed method benefits both management (by providing a truer appraisal of the project) and the developer (by providing the necessary technical details). We think the possibly higher early cost will serve to reduce later costs and prevent failures. And our concern is with the substance of the recommendation and not with "prior discovery," or name.

2.3.1 Applicability

This approach is, we believe, of general applicability (Fig. 14). Except in the formality of two contracts and in the possible competitive contractors aspects of the first contract, it represents simply a better organization of the work. We believe we have not added any unnecessary work or documentation and we believe that the more strict sequence of requirements, program design, program coding, and test is desirable in any project.

We recommend that it be first used in a high risk project. Generally, the risk is higher if the project is large, if the concepts are new, and if the purpose is "policy" as contrasted with "physical principles." Newness in hardware or system architecture can also contribute to higher risk.

If, then, it is successful in a high risk situation, we believe review will show that the success is for reasons which have more general applicability than just for high risk projects.

We recommend that this approach be used also when the software is only a part of a larger procurement, that the software part of that larger contract be treated in the two major phases we described.

If the software part of such a contract is considered a high risk, then perhaps it is a proper subject for an early separate contract (our first contract).

But it is most important to recognize that the importance of our two contract recommendation is not in two contracts, but rather in a well-defined first activity which creates an informed decision-to-

Figure 14

APPLICABILITY OF APPROACH

- **General**
  - **It organizes the work better**
  - **It delays development commitment until the true scope is known better**

- **First use**
  - **On high risk projects**

    (Large, new concepts, "Policy")
  - **Review success to confirm more general applicability**

- **As part of a larger procurement**
  - **If software is truly critical, early independent contract for feasibility and scope is required**

proceed point before commitment to the actual development. As such, the approach is recommended equally for stand-alone software and for software embedded in a total larger system.

2.3.2 The Detailed System Design Requirements Contract

As noted earlier, the fundamental purpose of this first contract is to put the actual development on a sound footing, to give the USAF an informed decision point before actually committing the development.

Figure 15 shows outputs we think important, although, as related to a specific program, the list may not be exhaustive or may (and we think this less likely) contain unnecessary items.

Detailed requirements, as noted, include the anomalous and extreme "missions" and collateral functions, as well as the primary mission.

Interoperability and interface details generally are usefully put in another volume. If other systems are involved, it is not enough simply to issue this document. The other systems must agree to its contents and, if needed, indicate what steps are required, etc., to meet the requirements.

User and maintainer plans and needs documents refer here to those things procured under the development contract. But not all plans and needs are developed or filled this way. Before development is authorized, the user and maintainer must complete their total plans, emphasizing the life cycle aspects, simply as part of sound management, and as necessary items for Management Review II.

Another highly important document is a description of how the hardware/software system will work as a whole. The idea is to start with the whole system and then devolve into more detailed descriptions of the parts. "Top-level" (which we use as a generic descriptor and not to indicate an endorsement of "top-down" design) design is there described -- how major sections of hardware and software interact -- to unify the detail. The contractually-oriented detail, the CPCI and CI descriptions, are interwoven in this, or given separately, as appears best, but the necessary breakouts are given. Obviously enough, if the Block I, II, III approach is to be used, the delineations are given here also. The description includes how growth and change are to be accommodated by fundamental design of the program and by accommodations of more memory or more processing capability.

And this description indicates how reliability and quality assurance are to be built in -- by hardware arrangements and by software structure. In this use of R&QA, we stress system availability for service, as well as "error-free" code production.

One collateral function, software for measurement of how the computer spends its time, is useful for ongoing activities in both growth and change and in R&QA assessment and improvement and ought to be a generally included function.

When operators (data input/output) are intimately involved, statistical data relative to them are also desirable. The need for such collateral data will extend through many systems and should be recognized

Figure 15

DETAILED SYSTEM DESIGN REQUIREMENTS CONTRACT

- After MANAGEMENT REVIEW I and approval of Program Management Directive (or equivalent).
- Produces detailed plans, specs, requirements.
  - Contents (all USAF property) include:
    - Detail requirement - primary, anomalous, extreme missions, collateral functions.
    - Interoperability and interface details (involve other systems as needed).
    - User plans and needs.
    - Maintainer plans and needs.
    - How the hardware/software system will work:
      - Includes top level design.
      - Includes CPCI and CI structure.
      - How growth and change accomodated
      - How reliability and quality are built in
      - CRISP and CPDP information integrated and coordinated with this

Figure 15A

DETAILED SYSTEM DESIGN REQUIREMENTS

Risk analysis and mitigation - simulation results for high risk items.

Updated system spec and PMD.

Development contract proposal(s):

- Money, schedule
- CPCI, CI definition
- Work breakdown structure
- Statement of work
- Development plans, tools, techniques
- Block I, II, III and "builds" definitions, when used
- Test Plan, needs
- USAF management and visibility structure.

• Enables an informed MANAGEMENT REVIEW II and development.

and included in the work.

The contents of this "how it works" document overlap, in part, with the current[10] Computer Resources Integrated Support Plan (CRISP), and with the Computer Program Development Plan (CPDP), which contain additional information closely related. It might be better to integrate the three into a new document, but since it would probably be too large, then we suggest divisions more along technical vs. contract/management lines.

Another document is devoted to high risk items within the software and to results of studies which show that the functions can be accomplished. Although final proof comes only when the actual code and computer are used, there are many ways of demonstrating feasibility without actual coding. Hardware/software tradeoffs may be required to accomplish some functions, which is why we recommend that such tradeoffs be made possible.

And, of course, this detail needs to be formalized into a specification (as a development contract reference point) and an updated Program Management Directive.

These documents are those taken to the Management Review II. They constitute the system definition. Together with them, the SPO can take the development proposals shown at the bottom of Fig. 15: cost and schedule (as proposals from each contractor) and the other items which describe the development contractually. These need not be identically the same for different (competing) study contractors. They describe how each contractor would go about meeting the system requirements in his own style.

2.3.3 The System Development Contract

Figure 16 touches again on contractual matters for the actual development. Each operational program should be a Computer Program Configuration Item (CPCI), with Block I, II, III deliveries adding up to one of the CPCIs called for in the contract. We call again for resident debug programs in this CPCI and a facility to use the outputs, the better to maintain the program.

Software tools for development, operations, and maintenance should be separate CIs with deliverables for users and for maintainers clearly distinguished.

"Blocks" are subdivided into "builds" or similar substructures and "builds" are the management tracking items. The flow of work and review proceeds in the usual fashion; that is, first the Preliminary Design is made for each build and reviewed, and then the design is completed and subjected to a Critical Design Review. Test plans flow simultaneously with these, with the tests including and even emphasizing the anomalous and extreme conditions as well as the primary mission.

Memory and timing budgets are used as management tools, as are tradeoffs between blocks, and hardware/software tradeoffs as well. The incentive fee, based on what is actually delivered and accepted by block, is a major tool in keeping to schedule and content in each block, with

Figure 16

SYSTEM DEVELOPMENT CONTRACT

- Each operational program is one CPCI, which may include Block I, II, III deliveries, for contractural simplicity and "system" focus.
  - Include a resident debug program in the operational program, and an analysis facility in the contract and life cycle.

- Tools, etc., for O&M (and development) are separate CPCIs.
  - Distinguish CIs for users and maintainer.
  - Include design, production, test tools as appropriate for life cycle.

- Develop and manage by "builds", or similar structure.
  - Focus on user capabilities which are demonstrable, distinguishable, separable in code.

Figure 16A

SYSTEM DEVELOPMENT CONTRACT (CONT.)

- "Accept" on block basis, with block documentation.
- Part II spec is the sum of the descriptions of the blocks.
- Require memory, timing budgets.

- Enable tradeoffs between blocks (time, money, capabilities, level of effort), with incentive fee as part of control.

- Enable more memory, other hardware/software tradeoffs to keep job moving.

- Keep user/maintainer involved.
  - Design reviews.
  - Adjustments in requirement details (formal only).
  - Use and evaluate blocks as delivered.

these tradeoffs exercised to mitigate problems. But it is still only a tool and not a substitute for informed and diligent management by the SPO.

Throughout the development, the user and maintainer must be kept fully involved in learning better what they will receive, in full and informed participation in design reviews and test activities, and in resolution of problems in requirements that may be uncovered. Change authorization should be only by a formal process and after proper deliberation.

The block delivery concept brings in the valuable feedback from actual use and evaluation of the early blocks while the development team is still available. We stress again the importance of treating the early blocks as usable and used items.

2.3.4 Implementation

We believe (Fig. 17) this approach will reduce the risk in development and will yield lower development and life cycle costs and a shorter development period. Cost and schedule should be much better known at the decision-to-proceed point.

The product should be better in several ways: in meeting realistic requirements which are current, in its interoperability and interfaces with other systems, in being better structured for use and for maintenance, and in having an early use and evaluation period which can yield feedback to the developers while the development contract is still in force.

To accomplish this (Fig. 18), AFR 800-14 needs revision because of the "two contract" concept, because new documents are recommended, and because the recommended user/maintainer organization participation must be by regulation and not voluntary or casual.

In revising 800-14, however, we do not mean to add new levels of detail and of specifications. Particularly, we recommend against a subordinate specification which details exactly one way to operate under our recommendations. We recognize that the flexibility we advocate makes for more choices and more decisions in how to manage a specific job, but we believe this to be for the good.

It appears our recommendation is compatible with DoD Directive 5000.29. Under 5000.29, the DoD operates a Software Management Steering Committee. We recommend the AFSC strengthen its support of USAF participation in this committee. To do this, and to coordinate and promote AFSC internal activities better, a stronger organization than the present XRF is needed. Specifically, the rank, staff size, and charter of the present XRF office all are insufficient.

Much was said during the Study about the differences between Regulation 300 (ADP) System procurements and 800 (Embedded Computers) System procurements. Since we did not review "300" methods in detail, it is difficult to make recommendations of greater specificity than that the subject needs work, that "300" approaches should be made to yield benefits, especially to Command and Control systems, and that

Figure 17

ADVANTAGES OF THIS STRUCTURE

- **Markedly reduces surprises and scrambling.**
- **First contract provides excellent decision-to-proceed point - promotes "design-to-cost" for development.**
- **Yields realism:**
  - **- In requirements and total system function.**
  - **- In life cycle aspects.**
  - **- In money and schedule for development.**
  - **- In interoperability and interfaces.**

Figure 17A

ADVANTAGES OF THIS STRUCTURE (CONT.)

- Keys system functions current.
  - Shorter time because orderly.
  - Continual user involvement.
  - Emphasizes life cycle upgrading, R&QA.
- Block I, II, III Delivery as one CPCI.
  - Reduces manpower peaking. Promotes continuity.
  - Emphasizes structure in design.
  - Permits early and orderly user evaluation and use.
  - Keeps expertise available during early use.
  - Makes adjustments easier contractually for USAF and contractor, but incentive fee encourages keeping to schedule.

Figure 18

TO IMPLEMENT THIS APPROACH

- Develop appropriate procurement and contract practices.
- Develop process for responsible user/maintainer involvement.
- Revise 800-14 appropriately:
  - Keep compatible with 5000.29.
  - Keep at same level as now.
  - This approach requires flexibility in details appropriate to needs of each job. Don't prevent it by creating new rigid specifications. Prepare to help SPOs make best use of the flexibility and get off to a good start.
- Cooperate with DOD Software Management Steering Committee. Appoint USAF focal point.
- Try it, track results, improve as appropriate.
- Work to improve coordination and cooperation under "300" and "800" directives.
  - Benefits of "300" need to become more available to "800" programs, and vice versa.

perhaps how to use "300" methods must be better explained. Quite possibly the reverse is also true, and "800" methods should be better understood by "300" users.

3.0 DEVELOPMENT PRACTICES AND TOOLS

Particularly because electronics technology is continually reducing the cost of computer hardware, the cost of software is now a major item in a system development. Although improvements in productivity have been achieved,[11,12,8,13,14,15] they have not matched the hardware cost improvements. This is partly because systems have become larger and thus more coordination between more people is required, making possible more misunderstandings, etc.

Computers also have become more powerful in their instruction sets. This is helpful when used properly, but often also makes possible several ways of programming a specific task. Thus, programming, in the final analysis, remains an intensely personal nonautomatic craft. Levels of individual productivity vary widely,[15] and assessment, verification, and clear documentation remain problems.

Nonetheless, improvements have been made, and methods of improving discipline and producitivy have been developed. Figure 19 shows some characteristics we think a good development organization will or should have.

Properly sequencing and completing development tools, operating systems, and applications code is good discipline. It will never be possible perfectly to separate these three. The next step will always discover flaws in the earlier steps and probably flaws in newly developed hardware. USAF contracts should allow for such in-use debugging of development tools and operating systems.

Higher Order Languages (HOLs) are the major ways productivity improvements have been achieved. They express more powerful concepts more succinctly and, by reducing the sheer quantity of code, also reduce errors.

For each HOL, a contractor (or the USAF Office of Primary Responsibility (OPR) for the HOL, see later) should keep a library of proven routines for reuse or at least for guidance for new versions, and should require that modules be small (only a few "ideas" in a module). Small modules are easier to work with in a day-to-day sense and are easier to understand, test, and document.[16]

The general tendency of a programmer is to write code as soon as possible. Good discipline says that the full design should be made before coding, that writing the code is not the time to develop the design. A major exception is in a risk item, especially when questions

Figure 19

DEVELOPMENT PRACTICES AND TOOLS

A GOOD CONTRACTOR WILL:

- **Complete development tools and operating systems before applications design and code needs them.**
- **Use a Higher Order Language for programming.**
  - **Use a library of proven routines.**
  - **Use small modules (100-200 statements or so).**
- **Complete design of each element before beginning coding.**
  - **Verify risk amelioration before proceeding.**
- **Emphasize structure in design.**
  - **Block I, II, III concept helps force this.**
  - **Top-Down good, not only way.**
  - **"Build" concept gaining use - oriented to user capabilities and functions.**

Figure 19A

DEVELOPMENT PRACTICES AND TOOLS (CONT.)

- **Enforce programming standards.**
  - **Structured programming widely adopted.**
- **Centralize data definition and description.**
  - **Create a dictionary for data with definers, users.**
- **Develop and use memory and timing budgets.**
- **Use an independent force to test at build level and above.**
  - **Stress, failure, capacity testing as well as against requirements.**
- **Use appropriate tools to promote efficiency and thoroughness.**
- **Have a management technique which promotes visibility, control.**

of sufficient computer power and speed are involved. Then it is necessary to design and write code or otherwise simulate[17] the process as early as possible, to assess alternatives, to make software/hardware tradeoffs, if appropriate, and, in general, to ensure against a later crisis. We consider such risk studies a mandatory part of the first contract.

Along with small modules, the proper current trend in software design is to give it better structure.[8,18] This makes development of the different parts by different people much more feasible; permits staggered deliveries (as our Block I, II, III), thus reducing peak demand for people and development facilities; makes for better test and maintenance; and is a method much to be preferred. Top-down design is currently well-publicized and has been effective, but is only one of several approaches, no one of which seems so superior that it should become the only way. The concept of a build,[18] for example, is also gaining advocates as a structure organization method. The important part of all these methods is clear function and interface definition and independency of modules as much as is appropriate.

Interfaces have always been problems in software, considering each software module as interfacing other modules through data interfaces. Some data are only one bit long, others may be voluminous. The definitions must be the same to all. To do this better, a new method gaining advocates is to make data definitions only once, have all users reference the common definition, and to have the HOL accommodate different types of data automatically. Extending this, a compiler can indicate the data used or set in a module (as a debug tool) and, in the ultimate, prepare a data "dictionary" (data definition typer, users, setters) for the whole program. We recommend this.

Since timing and memory are key elements in most software applications, we recommend development and management of budgets for these. Not inflexibly, but no control at all is simply sloppy management.

An independent test force is mandatory, we believe, in any sizable job. It guards against rationalization (rather than solution) of problems, and adds a different viewpoint. It is foolish, however, for testers to develop "secret" tests to spring on developers to make the tester look good. If a tester believes a program will fail under certain tests, that should be communicated as soon as possible to the developers. And we note that testing should include and even emphasize stressful situations ("more bugs are found at the boundaries") rather than the primary mission in normal conditions. It becomes difficult, of course, to conduct all possible tests. Part of the skill in test design is the selection of an effective group properly representing the full range of conditions.

To do all these things, tools are required and must not be neglected. They often are expensive. They are discussed in the following section.

Finally a good contractor will have a scheme for management visibility and control, which should suit his needs and the USAF's as well. And we believe an "open door" approach can be worked out to a proper level and is very desirable.

3.1 DEVELOPMENT TOOLS

Why specialized tools? The answer is the same for software as for all industry: to reduce cost and improve quality. The parallel continues: some jobs make expensive specialized tools worthwhile, smaller jobs can do with less.

In software, the trend toward tools is steady and is based on two motives. One is productivity, simply to reduce the amount of labor required. The other is completeness and correctness. Since software is so much a personal craft, the human traits of optimism, putting off consideration of details, distaste for documentation, and even laziness must all be countered. Management leadership, motivation, and discipline all count heavily, but it remains that much can be effectively automated through tools, and we find even small micro-processors, with small instruction sets, now being offered with sets of tools as a "natural" way of offering a product.

The applicability of tools extends from the initial conceptualization of a system to its use and maintenance, as Fig. 20 suggests. Reference 19 is an excellent discussion of tools.

To make requirements complete and specific, work is now under way on requirements analyzers,[20] which is discussed later, as being still in research.

For early analysis of risk elements (especially as to timing), a traditional path has been to design and code specific routines and run them. Most particularly, this requires either an emulator* or target computer at an early part of the development. Because neither may be available so early, and for many other reasons, this approach may be quite costly or even impossible. As such, some organizations[17] have developed functional and analytical methods to perform almost the equivalent analysis. (A few compiler systems support checkout at the Higher Order Language level, rather than target machine instruction level. This, too, is a step forward.) Such methods will be of value in some jobs, and their use (and sponsorship by the USAF) should be carefully considered.

Compilers and what we call "compiler systems" are now standard. Their purpose is to translate Higher Order Language statements into machine language code. We use "compiler systems" because the process should incorporate more functions, such as enforcing programming standards, checking for syntax errors, handling the various data definitions inherently and automatically, and building toward the data dictionary noted earlier. Further, the HOL statements, with comments added by the programmer writing the HOL statements (partially enforceable

---

* Several computers may participate in a development. The computer on which the developed computer program is to run is called the target computer. A different computer which can emulate or simulate the instruction set of the target machine is here called an emulator. The machine on which programs are compiled, etc., is here called the development computer.

Figure 20

TYPES OF TOOLS

- **Requirements analyzers.**
  - ***Still in research, should be pursued.***
- **Functional and analytical simulation.**
  - **In design and risk analysis**
- **Compilers and "Compiler systems"**
  - **Higher order language translators.**
  - **Enforce programming standards, syntax.**
  - **Use data dictionary automatically.**
  - **Automatic documentation, flowcharts (if needed).**
  - **Generate test data automatically.**
- **Debug.**
  - **Simulators at instruction level or at implementation language level**
  - **Trace capabilities in operational code.**
  - **Input data generators and results analyers for these.**

Figure 20A

TYPES OF TOOLS (CONT.)

- **Validation and Acceptance.**
  - **System development lab.**
    - **For software debug use.**
    - **"Real" hardware - prove hardware / software compatibility.**
    - **Delivered to maintainer if needed.**
- **Development management.**
  - **Status of work in progress easily collected.**
    - **Trouble report tracking.**
    - **Configuration management.**
    - **Change control.**
    - **Develop trend analysis as appropiate.**

by the compiler through "Standards" enforcement), may be of a deliverable quality as documentation, at least for use by the maintenance force, and may be automatically flowcharted, if flowcharts are appropriate. We cannot recommend for or against flowcharts as deliverable items in a contract. What we can say is that flowcharts made automatically from the delivered design and from the delivered code are probably the only accurate ones.

Finally, tools can generate sets of data which test the module in its possibly several modes of operation, i.e., these data input yield these data output. All of these things can be done by tools closely associated with the compiler, and we use "compiler system" to indicate the collection.

To test and debug a program, many alternative methods are used. For real time programs, simply because they are designed to run in real time and not at the same time to do detailed analysis of program flow, an emulator of the target machine is often used. This may truly be a simulator (run on a machine which uses a different instruction set) or may use a machine which can be microcoded to execute the instruction set of the target machine. The important distinction for debug purposes lies in the surrounding tools: easy ways to set up the test, insert data (e.g., imitating a keyboard operator), collect data on what program steps and data are executed, and organize and report intelligibly the output data. No one of these tasks is trivial, all are helpful and make for efficiency. As noted earlier, we recommend that operational programs include capability to help diagnose problems that turn up during use, how complex a capability depending on the operational program.

Such emulation of a program can generally only go so far and is used most often during module development. Often it simply costs too much to emulate an entire program, especially through many different tests. Then a systems lab becomes useful for debug of software, as well as for demonstration that the hardware, its input/output devices, and the interfaces all also work properly together. Again the scope depends on the job, but proof that hardware and software do work together must be mandatory before delivery acceptance.

As a final group of tools, it is now being realized[17,14,18] that a consolidated data base approach for the development can make several management functions easily implemented. Functions such as trouble report tracking, configuration management, change control, schedule keepings, and trend analyses based on these become much easier if all of the programs (and the tools) are kept in a single organized way and place, and their use (from original input of the HOL statements, through use of the "compiler system", through module test, through build and block assembly and test) easily controlled.

The problems with tools (Fig. 21) arise from several sources. Most of those discussed above are strongly associated with the Higher Order Language used, but the link includes debug tools as well. Thus, the more the number of HOLs, the more the needed work. We agree with the DoD's policy of freezing the number of HOLs to be used, not only because of the life cycle aspects in both personnel and facilities. The concept

Figure 21

PROBLEMS WITH TOOLS

- **Proliferation of higher order languages.**
  - **- Each spawns its own tools.**
  - **- DOD standardization good idea.**
  - **- DOD/USAF Office of Primary Responsibility (OPR) concept good.**
  - **- Should add "library of proven routines" function to OPR - keeper of tools associated with HOL.**
  - **- Appoint USAF focal point for OPRs.**
- **Cost**
  - **- Who pays - the application, the OPR, the maintainer who needs them - needs resolution.**
  - **- Should be separate items in contract for visibility and awareness.**
- **Transportability and reusability.**
  - **- Development computer generally not target computer.**
  - **- Contractor familiar with own tools, etc.**
  - **- GFE development computer cost saving may be illusory without good complete GFE tools.**

of an Office of Primary Responsibility for each HOL is also agreeable to us. But the function of "librarian of proven routines" is not a present part of OPR duties. We recommend it be added. It is true that only rarely are routines used again, because they can be improved, etc., but it is also true that even only a review of a proven routine will often prevent mistakes being made in a new version. The USAF focal point organization recommended earlier should include in its mission the role of USAF HOL OPR representation to DoD and act as interchange between OPRs.

Given a considerable maintenance phase for a system, tools are more easily justified as part of the development contract. Who pays (or which account is charged) can remain a problem, intensified when the maintenance phase is small and the development must bear the cost. Certainly the cost should be clearly separate contractually. It is not clear to us, however, who should pay and what the USAF should do in a competitive situation when one contractor has tools already available and the other does not. One approach, having HOL OPR function as a collector (from developments) and/or processor of tools, does seem to have merit and we recommend its consideration.

In this, the USAF must recognize also that the implementations of a HOL and the tools that go with it will be different for each development computer. Further, each contractor will become familiar with a specific set and will probably suggest modifications and enhancements.

Faced with such complexity and with the cost of development computers, it has been suggested that the target machine also be the development machine, and the machine and the tools all be Government Furnished Equipment to the contractor. Such cost savings may be illusory. The machine must have the required capability, the tools must be available, and the USAF must have a way to discharge its responsibility for the tools and for the hardware during the development. This approach is not a panacea, simply another alternative.

Thus, one part of the problem is that the issues have not yet been worked on enough. They are important: good government policy can lead to improved development effectiveness. The AFSC needs to develop its own informed opinions and coordinate them with the DoD through the Software Management Steering Committee using the focal point organization recommended earlier.

4.0 VALIDATION AND VERIFICATION (V&V)

Two major and different activities are both called Validation and Verification (Fig. 22).

One is more properly called System Engineering. Since we believe that the true function of a SPO is to deliver an operational system (and its associated support systems) well-suited for its life cycle, it follows that we also believe the SPO is responsible for Systems Engineering. Validation and verification of requirements and of software design is one of a System Engineer's functions, while resolution of tradeoffs within a system, e.g., hardware/software is another System Engineering function.

Staffing the SPO System Engineering function can be done using USAF personnel, Federal Contract Research Center personnel, or an independent contractor not involved in the development. Whichever is used, the activity must be properly staffed from the beginning of the first contract, since the first contract is intensely a system engineering task. The function must continue through the development contract to ensure the cohesiveness of the system through development.

This is not to say there should be no system engineering function or responsibility placed on the design contractor(s) or on the development contractor. It is quite important that there should be. Most important is the case of a prime contractor, when the computer system is fully embedded, but in all cases, the contractor will best know what capabilities are being provided. System engineering responsibility then places discipline to ensure appropriate performance.

What we consider validation and verification is ensuring that the system works and meets the requirements. The development contractor is the responsible party, of course, but it appears almost always desirable to have an independent force act also. An independent force within the development contractor is often appropriate, but other contractors may also be used, depending on the scope and criticality of the job.

This activity is not simply a tail-end testing of the code to be delivered. It is entirely too artificial to look only at the requirements and the specifications to design the tests. Such tests are only the first of several types. More fruitful is stress, failure, and capacity testing -- when conditions are extreme or anomalous, or when part of the system has failed, or as a measure of how much activity can be handled (particularly in Command and Control systems) and what the

Figure 22

VALIDATION AND VERIFICATION

- **USAF system engineering capability in SPO is valuable, recommended.**
  - **From beginning of first contract.**
  - **USAF, Federal Contract Research Center , or independent contractor personnel as appropriate.**
- **V&V is insuring the computer system works:**
  - **By analyzing the design.**
  - **By reading the code (most effective).**
  - **By testing - stress, failure, capacity, requirements.**
  - **By development or independent contractor, or user, or maintainer, as appropriate to job risk, value, life cycle activities.**

  Must be involved from development contract on.

reaction of the system is to such conditions.

To devise good tests, the test designer needs to know how the system is designed. In doing so, he can accomplish useful debug work as well as plan the tests. Many studies[8,9] show that most errors can be discovered simply by analyzing the design and reading the code. Simulations and/or function modeling is useful for both requirements validation and design validation. As such, Reliability and Quality Assurance (R&QA) includes auditing the design and reading of the code as well as testing the product. How well a contractor understands how to build reliable software is one criterion for contractor selection, but an audit function during design, as well as System Engineering studies of R&QA aspects of design, are both needed.

Thus, the important point in our endorsement of System Engineering and Validation and Verification activities is that both begin early, System Engineering beginning and ending with the SPO itself, and Validation and Verification beginning with the development and continuing through the development.

5.0 PERSONNEL

Our major recommendation about personnel is that assignment to and detachment from a SPO should be related to the SPO's work rather than a fixed number of years, etc., as might be related to career paths for officers. Despite regulations and "standard" procedures, each individual has a style of management and continuity that is valuable. Possibly more important is establishing accountability. Too many management changes mean that no one can ever be held fully accountable, and no one person ever gets to have responsibility from the initial organization of the SPO to the final delivery of the product.

We recognize, of course, that such a principle can never be rigidly enforced, that there are many exceptional reasons for changes in personnel during a project. But changes should be exceptional, we recommend, rather than a standard practice.

Exceptional reasons apply more validly for senior officers than for junior officers. We recommend that junior officers remain in their SPO assignments through a full cycle of development so they can see the later consequences of earlier actions and get a rounded perspective of management. And this suggests that the table of organization of a SPO, especially the ranks assigned for different positions, be flexible as to rank. Promotion of a junior officer should not automatically require a transfer.

Such full job cycle training of junior officers is a long-term step toward creating an improved and experienced software management cadre. For the more pressing short-term problems, because the USAF does not now have widespread expertise in software management we recommend additional training courses be set up for both pre-duty (in a SPO) and continuing training purposes. Since the topic is management-oriented, short courses are needed not only in regulations and practices such as 800-14 and the Acquisition Management Guidebook Programs (ESD, ASD) based on 800-14, but also in case histories and lessons learned in real procurements. Theoretical management practices can easily be boring without "reasons why" being given, and case histories are a very effective way to bring out the "reasons why."

Within the DoD Software Management Steering Committee is a task to promote appropriate personal education. We recommend USAF support of this

Figure 23

PERSONNEL

- **Tour of Duty must be related to job.**
  - **- Continuity of direction**
  - **- Accountability**
  - **- Full cycle OJT for junior officers**
- **Short training courses needed now.**
  - **- Pre-duty and continuing education**
  - **- Add case histories of projects, how big systems work to usual computer science topics**
- **Need for specialists in compilers, etc, is small, as for HOL OPR's. Major need is for mature knowledge of how to use computers, how systems work, as in systems engineering.**

Figure 23A

PERSONNEL (CONT.)

- **Career concept still evolving**
  - **- Need role model for success**
  - **- Career path for 51XX in weapon systems is still unclear.**
  - **- Personnel system with 3 AFSCs and 3 SEIs per person, followed up with resumes and interviews seems adequate.**
  - **- Present concentration of expertise, as with MCI at ESD, effective under present conditions. Other AFSC organizations might well institute similar offices for the next few years. But the long-range goal should be to reduce or eliminate these offices by creating more ubiquitous capability.**

activity (again using the focal point organization) with the caveat that the need within the USAF for professional computer scientists is different from the need for management expertise. Management requires technical understanding, but more in a system engineering sense than in a how-to-program sense. HOL OPRs, for example, do require even PhD level training in computer science topics. SPOs do not.

The career concept for software management "specialists" seems to us still not clear. For example, there are few officers at general officer rank whose careers have involved software in any major way. Establishment of the 51XX Specialty Code series is a right initial step, but our discussions brought out that it may be more associated mentally with Automatic Data Processing careers than Weapon System Management careers. The two are similar in some ways, but different in others. We do not recommend an alternative, but rather high level demonstration that the USAF considers 51XX a valid, important Weapons Systems Management specialty.

As to selection of personnel for a SPO, we believe the present semi-automated process can be made to work. Using three Specialty Codes and three Special Experience Identifiers per person does permit an individual to represent himself reasonably well in such a system. A resume is also well nigh mandatory, and interviews ought also be encouraged as part of the personnel selection for a SPO process.

Because of the lack of well-experienced personnel, ESD has created the Deputy for Surveillance Division (MCI) office as a concentration of available experts. As an expedient measure, this technique will work, and other AFSC organizations should consider seriously whether the approach would serve them well also. But still as an expediency, for only a limited numbers of years. The long range goal must be to make such offices unnecessary by increasing the general competence in software management within the USAF.

6.0 TECHNOLOGY BASE

Computer hardware and software is perhaps the most rapidly changing technical discipline. Many people agree that a "digital revolution" is in process, similar in importance to the "transistor revolution" of the 40s and 50s.

Under such rapid change, the ability to manage and the ability to make most effective use of the new technology generally suffer. Such is the case with software. Technology base activities need, therefore, to improve the present as well as plan for the future.

All aspects of current software can be improved (Fig. 24). For management enhancement, better ways of predicting the size of a program are needed, which is part of better ways of predicting the cost of development. Another part of cost estimation is productivity. There is not a universal standard way of calculating (or defining) productivity nor of saying what truly affects it. Nor is there a sure way to distinguish high quality programs from low quality, except by test. All of these matters need continued research to find out what is happening today to quantify the relationships, and thus to know the higher payoff areas for improvement. The data gathering part of this work needs to be written into development contracts, given approved definitions of the data to be collected.

Part of this data gathering and a part of contract management is the Work Breakdown Structure (WBS) specified in the contract. To our understanding, it is the accounting counterpart of our blocks, builds, and modules. With it goes the cost schedule control system[21] criteria used now as part of management assessment of progress.

Work on these is needed. We believe our two-contract approach will lead to cost and schedule success in development, in part because the development can be better organized and make tools like the WBS and the cost schedule control system more effective because they reflect the actual work more accurately. And we believe this better organization of the work will make its true progress more visible. We understand the often great anxiety of development managers as to whether or not all of the activity constitutes progress.

Part of the solution is greater technical competence in the managers, but better measures of productivity and quality, better organization of the work, and better WBS and other management tools, as well as better work organization, will all contribute to greater effectivity.

Figure 24

TECHNOLOGY BASE PROGRAMS

- **All aspects of current software can be improved, notably:**
  - **Metrics for sizing, costing, productivity, quality**
  - **Work breakdown structure**
  - **Cost-schedule control system criteria**
  - **Requirements analyzer**
  - ***Improved higher order language* (HOL)**
  - **Development tools (use HOL OPR)**

    Scope, certification, transferability, transportability
  - **Library of proven routines (use HOL OPR)**
  - **Design methodology - better structure to promote understanding, documentability, maintainability, etc.**
  - **Flexible instruction set computers for operations and for simulation (debug and lab)**

Figure 24A

TECHNOLOGY BASE PROGRAMS (CONT.)

- **Toward the near future**
  - **How best to use, manage, etc., mini-and microprocessors, distributed computing and data bases**
  - **Data transfer for multi-system interoperability, especially interservice**
    - **Hardware, protocols, formats**
    - **Common Language descriptors**
    - **Easy input/output**
- **GFE software production centers - need first solve:**
  - **Security**
  - **Priority, size of facility**
  - **SPO control vs. center control**
  - **Accountability of USAF**
  - **Life cycle sufficiency**
  - **Distributed vs. concentrated**
- **Cooperate with DOD Software Management Steering Committee. Appoint USAF focal point.**

During the Study, one presentation[20] was on a Computer Aided Requirement Analyzer (CARA) approach to ensuring that requirements were consistent and complete. We recommend this type of work be pursued, but we consider it still in a research/early prototype phase rather than ready for general use. Of particular concern is that the need for precision language that is common in working with computers is present here also, and possibly to a level that normal management cannot tolerate. We believe the designers of CARA understand that its users shold not be computer programming types, but we do not believe the present structure meets those needs. More work is also needed to make requirements validation better. Simulation and/or functional modeling, now used more for debug of design, can be a tool for requirements analysis also, and should be so extended.

The DoD has frozen the number of Higher Order Languages (HOLs) to be used within the DoD. This is good, in warding against needless proliferation. But DoD is also working toward a better HOL, which we support. Improvement should not be sacrificed to standardization, but rather controlled.

Within this activity is the appointment of Offices of Primary Responsibility for HOLs. Scope, certification, transferability, and transportability of HOLs and the tools they spawn are appropriate duties of OPRs. To this we recommend adding a librarian function for proven routines, as noted earlier.

Research should also be pursued in design methodology, aiming for better understanding, documentability, maintainability, etc.

Research should also be pursued in flexible instruction set machines. Thus, code written for one machine could be run on another (with greater power, for example). Or the flexible machine could be used in a lab where its greater power would permit extra information, etc., to be gathered during testing.

We believe mini- and microcomputers are successful in many current applications but the tradeoffs are not always well understood. We recommend the USAF develop guidelines, but not rigid directives, through study, for their use. The USAF should consider such topics as logistics and maintenance as well as how well they compute. What is good for a one-installation system may not be right for a multi-installation system.

The interface problem needs special attention, we believe, at all levels. That is, for data communication, the USAF could benefit from standardization in hardware, protocols, and formats. This applies to moving data from one point to another. At a higher level, the data itself would benefit from standardization in symbology and meaning. Especially this is true in interservice tactical systems and Command, Control and Communication (C^3) systems. A relevant project is the Bell System's Common Language system, in which, at considerable dollar and time costs, the Bell System developed a standardized symbology for all its apparatus and equipment. Particularly in interservice matters, DoD lack of such standard nomenclature appears to be the current case and inhibits the total effectiveness of tactical and C^3 systems.

One result of the lack of Common Language is that entry of data into systems becomes texual and tedious. None of these systems is useful if

data are not faithfully and accurately entered. And so we also recommend continued work on easy input and output data methods.

Discussed during the Study was the Defense Advanced Research Projects Agency (DARPA) National Software Works concept (using the DARPA network as a research vehicle). The thought is that different computers, different people, etc., do different things best and, if this individual expertise can be both kept and unified into a system (through networking, for example), then great effectiveness should result. Somewhat differently, the system could be made available at a single location, but made "universally" available through networking. Technically, the Study Group believes this feasible. Administratively, there are many problems which need work: security, priority (which involves size of the facility so that user demands can be met), administrative control (competition between SPOs and internal organizational responsibility), accountability of the USAF to the contractor (e.g., for a HOL compiler which causes subtle errors not apparent for some time), and life cycle sufficiency. In fact, we believe the technical issue of distributed vs. concentrated facilities to be of less difficulty than the administrative matters. The concept has appeal in avoidance of duplication and in concentration of expertise: we recommend the administrative topics be studied thoroughly and promptly.

Because of its potential scope, this issue needs to be addressed both for the AFSC (and the USAF), and for the DoD, the latter through the DoD Software Management Steering Committee.

7.0 REFERENCES

1. DeRoze, B. C., "Defense System Software Management Program Overview." *(Presentation to the Summer Study)*

2. Manley, J. H. and Lipon, M., "Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (SRWG Report)." Two volumes, NTIS Documents AD-A018881 (2), November, 1975.

3. Everett, R. R., "Some Comments on Software," Mitre Corporation, December, 1975.

4. DeRoze, B. C. "Defense System Software Management Plan," NTIS Document AD-A022 558, March, 1976.

5. Anon., "Lessons Learned - Computer Hardware/Software Acquisition," Hq ESD/MCI, April 1, 1976.

6. Anon., "Project Pacer Flash - Executive Summary and Final Report," AF Logistics Command/MMO, September, 1973

7. Anon., "A Case Study and Evaluation of the Combat Grande Software Acquisition," AFSC/ESD/MCI, February, 1976.

8. Mangold, E., "Software Testing - Validation and Verification." *(Presentation to the Summer Study)*

9. Boehm, B. W., "Software Reliability: Measurement and Management." *(Presentation to the Summer Study)*

10. Anon., "AF Regulation 800-14 Volumes I and II - Management of Computer Resources in Systems," September, 1975.

11. Parten, R. P., "Space Shuttle Orbiter Avionics Software Management." *(Presentation to the Summer Study)*

12. Glaser, P. F., "Citicorp Bank-Consumer-Retail Network." *(Presentation to the Summer Study)*

13. Crowley, T. H., "The Safeguard System Software Testing."
(Presentation to the Summer Study)

14. Royer, R. D., "Software Development of No. 4 Electronic Switching System."
(Presentation to the Summer Study)

15. Case, R. P., "Programming Quality and Productivity."
(Presentation to the Summer Study)

16. Ingrassia, F. S., "The Unit Development Folder - An Effective Management Tool for Software Development," TRW Defense and Space Systems.

17. Drane, L. W., "Some Software Tools Used to Develop other Software."
(Presentation to the Summer Study)

18. Munson, J. D., "The System Development Corporation Software Factory*." (* Trademark of SDC)
(Presentation to the Summer Study)

19. Reifer, D. J., "Automated Aids for Reliable Software," IEEE International Conference on Reliable Software, April, 1975.

20. Melanson, A., "Computer Aided Requirements Analysis (CARA)."
(Presentation to the Summer Study)

21. Anon., "The Cost/Schedule Control Systems Criteria," in DoD Directive 7000.2 - "Performance Measurements for Selected Acquisitions," December, 1974.

BIBLIOGRAPHY

Anon., "Department of Defense Directive 5000.29 - Management of Computer Resources in Major Defense Systems," Department of Defense, April, 1976.

Belady, L. A., and Lehmann, M. M., "The Evolution Dynamics of Large Programs," IBM System Journal, September, 1976.

Bratman, H., and Court, T., "Elements of the Software Factory," System Development Corporation, SP-3851, 1976.

Carlson, W. E., "Software Research in the Department of Defense, Defense Advanced Research Projects Agency, Department of Defense, 1976.

Cerf, V. G., "Observations and Recommendations for Research and Development in Support of Software Production and Management," August, 1976. *(Unpublished)*

Jones, M. N. "HIPO for Developing Specifications," Datamation, March, 1976.

PRESENTATIONS

Belady, L. A., "The Evolution Dynamics of Large Programs." (Co-author, M. M. Lehman.)

Boehm, B. W., "Software Reliability: Measurement and Management."

Canaday, J. J., "The B-1 Program, Software, and Avionics Software."

Carlson, W. E., "Defense Advanced Research Projects Agency - Information Processing Techniques Office Projects - National Software Works."

Case, R. P., "Programming Quality and Productivity."

Cerf, V. G., "Computer Networking."

Crowley, T. H., "The Safeguard System Software Testing."

DeRoze, B. C., "Defense System Software Management Program Overview."

Drane, L. W., "Some Software Tools Used to Develop Other Software."

Duff, T. O., "The 485L Program Software."

Edge, R. L., "Mission of USAF Assistant Chief of Staff/Computer and Communications Resources Office."

Emma, F. T., "ADP System Software Management Program Overview."

Funkhauser, V., "Hq AFSC Program Management Assistance Group (PMAG)."

Funkhauser, W., "PMAG Charter and Method of Operation."

Glaser, P. F., "Citicorp Bank-Consumer-Retail Network."

Gordon, S. C., "AFR 800-14 Volumes I and II: Computer Resources in Systems."

Gordon, S. C., "The Management of Computer Resources in Systems."

Holtz, D. C., "Example of B-1 Avionics Software Development."

Jarrell, T. H., "Personnel Practices."

Mangold, E., "Software Testing - Validation and Verification."

Melanson, A., "Computer Aided Requirements Analysis (CARA)."

Munson, J. D., "The System Development Corporation Software Factory."*
(* Trademark of SDC.)

Parten, R. P., "Space Shuttle Orbiter Avionics Software Management."

Royer, R. D., "Software Development of No. 4 Electronic Switching System."